(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
27 octobre 2005 (27.10.2005)

PCT

(10) Numéro de publication internationale
WO 2005/101725 A1

(51) Classification internationale des brevets[7] : H04L 9/32, G07F 7/10, G06F 1/00

(21) Numéro de la demande internationale :
PCT/EP2005/050828

(22) Date de dépôt international :
25 février 2005 (25.02.2005)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
0450553 19 mars 2004 (19.03.2004) FR

(71) Déposant *(pour tous les États désignés sauf US)* : GEMPLUS [FR/FR]; Avenue du Pic de Bertagne Parc, d'activité de Gémenos, F-13420 GEMENOS (FR).

(72) Inventeurs; et
(75) Inventeurs/Déposants *(pour US seulement)* : CHEVALLIER-MAMES, Benoit [FR/FR]; La Sardanne Les Brayes, F-13260 CASSIS (FR). NACCACHE, David [FR/FR]; 52 rue Letort, F-75018 PARIS (FR). PAILLIER, Pascal [FR/FR]; 37 cours de Vincennes, F-75020 PARIS (FR).

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

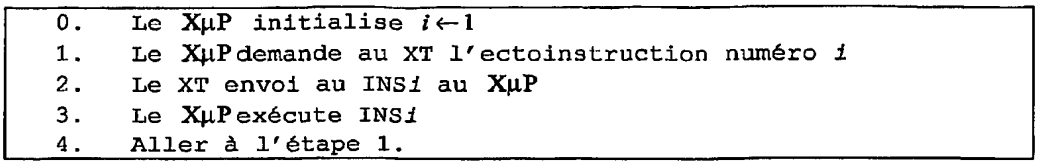
(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :
— *avec rapport de recherche internationale*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

(54) Title: METHOD FOR DYNAMICALLY AUTHENTICATING PROGRAMMES WITH AN ELECTRONIC PORTABLE OBJECT

(54) Titre : PROCEDE D'AUTHENTIFICATION DYNAMIQUE DE PROGRAMMES PAR UN OBJET PORTABLE ELECTRONIQUE



(57) **Abstract:** The invention concerns a method for dynamically authenticating an executable programme, that is the continuation of the instructions defined thereby. More specifically, the authentication of a programme is performed repeatedly during the very execution of said programme. The method for making secure an electronic portable object through execution of a programme P supplied by another insecure electronic object uses, inter alia, a secret key protocol.

(57) **Abrégé :** La présente invention décrit un procédé permettant d'authentifier dynamiquement le contenu d'un programme exécutable, c'est-à-dire la suite des instructions que celui-ci définit. Plus précisément, l'authentification d'un programme est réalisée de manière répétée au cours de l'exécution même dudit programme. Le procédé de sécurisation d'un objet portable électronique exécutant un programme P fourni par un autre objet électronique non sûr, utilise, entre autre, un protocole à clé secrète.

WO 2005/101725 A1

PROCEDE D'AUTHENTIFICATION DYNAMIQUE DE PROGRAMMES PAR UN OBJET PORTABLE ELECTRONIQUE

Ce brevet d'invention décrit un procédé permettant d'authentifier dynamiquement le contenu d'un programme exécutable, c'est-à-dire la suite des instructions que celui-ci définit. Plus précisément, l'authentification d'un programme est réalisée de manière répétée au cours de l'exécution même dudit programme.

Le principe de fonctionnement de l'invention permet de concevoir un nouveau type d'élément sécurisé appelé « Externalized Microprocessor » où XμP qui, contrairement à d'autres dispositifs de calcul tels que la carte à puce (objet de nombreux brevets comme par exemple FR.2 266 222) ne contient pas de mémoire programme (classiquement appelée mémoire ROM, de l'anglais « Read Only Memory »). A la différence des dispositifs classiques, en effet, le XμP peut exécuter des programmes qui lui sont transmis au moment même de leur exécution, en toute sécurité.

Les avantages qu'un dispositif mobile de calcul sans mémoire ROM présente par rapport aux technologies de calcul embarqué classiques (nous prendrons la carte à puce comme technologie de référence) sont extrêmes :

- le masquage, opération industrielle au cours de laquelle on grave une mémoire ROM spécifique, disparaît totalement;

- la correction des bogues se résume à une mise à jour des programmes stockés dans le disque dur des terminaux ou sur un réseau de communication tel qu'Internet, et ne nécessite donc pas de devoir

retirer du marché ou renouveler des cartes à puce défectueuses ;

plus important encore, la taille des programmes n'est plus un facteur limitant.

Ce dernier avantage est d'autant plus attractif que la tendance technologique a toujours été de faire exécuter à la carte à puce des programmes de plus en plus complexes et donc de plus en plus volumineux. D'un point de vue industriel et fonctionnel, une carte à puce est un ordinateur miniature. Une petite mémoire volatile RAM (de l'anglais « Random Access Memory ») embarquée avec le microprocesseur sert à stocker les résultats temporaires d'un calcul et le microprocesseur de la puce exécute un programme écrit de manière non modifiable dans la mémoire ROM : l'homme de métier emploie le terme de gravure, cette gravure ayant été effectuée à l'étape dite de masquage. Ce programme ne peut ensuite plus être modifié de quelque façon.

Pour le stockage de données spécifiques à l'utilisateur, les puces contiennent une mémoire non volatile EEPROM (pour « Electrically Erasable and Programmable ROM ») ou Flash, ces deux types de mémoires pouvant autoriser à la fois lectures et écritures par centaines de milliers. Les cartes Java, cartes à puce particulières, permettent même l'importation de programmes exécutables (appelés « applets », acronyme de l'anglais) dans leur mémoire non volatile selon les besoins du détenteur de la carte. En outre, les cartes Java de dernière génération embarquent un éditeur de lien (« linker »), un module de chargement (« loader »), une machine virtuelle Java, des modules d'appels de méthode à distance (« remote method invocation module» en anglais),

un vérificateur d'applet (« bytecode verifier»), un pare-feu pour applications Java résidentes (« applet firewall »), un ramasse-miettes (« garbage collector »), des librairies cryptographiques, des gestionnaires complexes de pile, etc.

Finalement, une carte à puce comprend un port de communication pour l'échange de données et d'information de contrôle avec le monde externe. Un taux de communication classique est de 9,600 bits par seconde, mais des taux beaucoup plus rapides compatibles avec la norme définie par l'ISO (« International Organization for Standardization ») sont généralement employés (de 19,200 jusqu'à 115,200 bits par seconde). L'apparition du protocole USB dans le monde de la carte à puce ouvre de nouveaux horizons et permet facilement d'atteindre des débits de l'ordre du mégabit par seconde. Dans ce contexte, il devient tentant d'extraire la mémoire ROM du modèle de fonctionnement de la carte à puce, et de s'appuyer sur un protocole de communication ultra-rapide pour transmettre lorsque nécessaire les programmes qu'elle contenait auparavant.

D'un autre côté, faire exécuter par un dispositif mobile un programme exécutable transmis par un terminal potentiellement non-sûr et malveillant pose d'importants problèmes de sécurité. Le problème essentiel d'une telle approche réside dans la présence de clés cryptographiques stockées dans la mémoire du dispositif lui-même. Un programme malveillant (distinct par voie de conséquence des programmes s'exécutant légitimement sur le dispositif) pourrait en effet tenter de révéler ou modifier la valeur desdites clés, invalidant ainsi totalement la sécurité des applications les utilisant pour fonctionner.

L'invention que nous allons décrire permet de répondre très efficacement à cette problématique avec l'aide de

fonctions de cryptographie symétrique (dite aussi cryptographie à clé secrète) classiques et efficaces : une fonction de MAC (selon l'acronyme anglais « Message Authentification code ») et quelques fonctions de hachage, traduction du terme anglais « hashing » provenant du verbe « to hash ».

Ces fonctions de hachage seront notées $HASH_1$, $HASH_2$ et $HASH_3$ dans le brevet. Conformément à l'état de l'art, ces fonctions sont définies par une fonction dite de compression. Par définition, on dit que HASH est une fonction de hachage définie par une fonction de compression H et par une constante IV (de l'anglais « initialization vector »), lorsque la définition suivante s'applique:

$$HASH(a_1, a_2, \ldots, a_k) = H(HASH(a_1, \ldots, a_{k-1}), a_k)$$

avec le cas particulier suivant :

$$HASH(a_1) = H(IV, a_1)$$

les nombres entiers a_1, a_2, ..., a_k désignant ici les arguments de la fonction de hachage.

Dans ce document, nous utilisons donc les fonctions de hachage $HASH_1$, $HASH_2$ et $HASH_3$ qui sont respectivement définies par (H_1, IV_1), (H_2, IV_2) et (H_3, IV_3).

Ainsi, le résultat d'un hachage se calcule itérativement à l'aide d'une boucle et plusieurs appels à la fonction de compression déterminant le hachage. De telles fonctions de hachage sont très classiques en cryptographie: citons par exemple les fonctions de hachage SHA et MD5 dont les spécifications reposent sur la description donnée ci-dessus.

La présente invention sera plus facilement comprise à l'aide des figures jointes.

La Figure 1 décrit la sémantique dynamique d'un exemple de jeu d'instructions appelé XJVML permettant

d'illustrer de façon non limitative les différents modes de réalisation de l'invention.

La Figure 2 décrit le procédé naïf de l'état de l'art permettant, de façon non sûre, d'exécuter un programme P fournit par le monde extérieur au XμP.

La Figure 3 décrit une politique de sécurité en XJVML selon l'invention, autorisant la lecture et l'écriture de données dites publiques.

La Figure 4 décrit une politique de sécurité en XJVML selon l'invention, autorisant uniquement la lecture de données dites publiques.

La Figure 5 explique la gestion de la politique de sécurité au cours de l'exécution du programme P.

Dans la suite du texte, nous verrons un programme donné P défini sur un jeu d'instruction (ou langage de programmation) comme une suite ordonnée d'instructions :

1 : INS_1
2 : INS_2
3 ...
F : INS_F,

ces instructions étant positionnées à des adresses appartenant à l'ensemble {1,...,F}, F désignant le nombre d'instructions du programme P.

Nous définissons également, à titre d'exemple illustratif non limitatif, un jeu d'instruction appelé XJVML qui servira à illustrer les modes de réalisation de l'invention.

XJVML décrit une architecture simpliste basée sur le processeur virtuel JVML0 défini dans le document de R. Stata et M. Abadi intitulé en langue anglaise « *A type System for java Bytecode Subroutines* » publié dans le

document référencé SRC Research Report 158 le 11/06/1998 disponible à l'adresse électronique suivante :

http://www.research.digital.com/SRC/.

L'architecture sur laquelle opère XJVML est semblable au modèle de calcul connu de l'homme de l'art comme étant celui de von Neumann, à ceci près qu'elle ne contient pas de mémoire programme. L'architecture de XJVML comporte :

- une mémoire volatile appelée RAM,
- une mémoire non volatile appelée NVM,
- un générateur de nombre aléatoire appelé RNG,
- une pile d'opérande appelée ST,
- un port de communication (dit aussi d'entrée/ sortie) appelé IO.

Le jeu d'instructions de XJVML est défini par les instructions suivantes, où x dénote une donnée immédiate, L est l'adresse d'une instruction avec $1 \leq L \leq F$ et F est le nombre d'instruction du programme considéré :

- L'instruction 'inc' incrémente la donnée se trouvant sur le sommet de la pile. L'instruction 'pop' retire l'élément de pile se trouvant à son sommet : on utilisera le vocable « dépiler ». L'instruction 'push0' ajoute la donnée constante 0 au-dessus de l'élément se trouvant au sommet de la pile : on utilisera le vocable « empiler ».
- L'instruction 'load x' empile la donnée se trouvant à l'adresse x en RAM. L'instruction 'store x' dépile la donnée au sommet de la pile et la recopie à l'adresse x en RAM. L'instruction 'load IO' capture la donnée présentée sur le port de communication et

l'empile tandis que l'instruction 'store IO' dépile la donnée supérieure de la pile et la recopie sur le port IO. L'instruction 'load RNG' produit un nombre aléatoire et l'empile. L'instruction 'store RNG' n'existe pas.

- L'instruction 'if L' observe la donnée au sommet de la pile et initialise le compteur de programme à L si cette donnée n'est pas nulle.
- L'instruction 'halt' arrête l'exécution du programme.
- L'instruction 'getstatic x' empile la donnée stockée en NVM à l'adresse x et l'instruction 'putstatic x' dépile la donnée supérieure de la pile et la stocke dans la mémoire non volatile à l'adresse x.
- L'instruction 'xor' dépile les deux données supérieures de la pile, calcule le XOR (OU EXCLUSIF) de ces données et empile le résultat. L'effet de l'instruction 'dec' est l'exact opposé de celui de l'instruction 'inc', c'est à dire que la donnée supérieure est décrémentée de 1. L'instruction 'mul' dépile les deux données supérieures, les multiplie et empile les deux données représentant le résultat sous forme de deux mots, l'un de poids fort, l'autre de poids faible. L'instruction 'goto L' est un simple saut à l'adresse de programme L. Enfin, l'instruction 'div' dépile les deux données supérieures, divise la moins haute de ces deux données (le numérateur) par la donnée la plus haute dans la pile (le dénominateur), et empile la donnée résultant de l'évaluation du quotient. Il est à noter que si, pour l'instruction 'div',

le dénominateur est nul, une exception est exécutée, et le compteur de programme est réinitialisé à l'adresse du début de l'exception, adresse appelée AdExcDiv par la suite. Cette exception est appelée l'exception « division par zéro ».

La sémantique dynamique de notre jeu d'instructions est schématisée à la Figure 1 (à noter qu'il n'y a aucune règle pour l'instruction 'halt'. Dans la Figure 1, « **undef** » désigne la donnée par défaut d'une cellule de la mémoire.

Il est implicite que les instructions qui utilisent la pile provoquent une interruption si la pile est vide, c'est à dire, en dénotant par s le nombre d'éléments de la pile, si s = 0, ou bien si elle contient insuffisamment de données, par exemple lors de l'exécution d'une instruction 'xor' alors que s = 1.

On rappelle que le terme XµP désigne le dispositif soumis au procédé de l'invention, c'est à dire un dispositif électronique dépourvu de mémoire programme, et que similairement, le terme XT désigne l'« Externalized Terminal », c'est à dire le terminal qui communique avec le XµP et contient le programme P que celui-ci exécute.

On rappelle aussi que le programme P introduit dans chaque terminal XT (que l'on rappelle être non sûr et possiblement malveillant) se présente sous la forme d'une suite d'instructions :

1 : INS_1
2 : INS_2
3 …
F : INS_F

Le principe de l'échange entre le XμP et le XT est très simple: lorsque l'exécution commence, le XμP initialise à 1 son compteur de programme, référencé ci-dessous par la variable i, et demande l'instruction d'adresse i au XT. Le XμP exécute INS_i, mettant ainsi à jour son état interne et déterminant par conséquent la nouvelle valeur du compteur de programme. Le compteur de programme i et l'adresse de INS_i se confondent lors de l'exécution du programme. Ainsi, lors de l'exécution du programme, i désignera de façon égale l'adresse comme le compteur de programme. Ce procédé est répété tant que l'instruction de fin de programme n'est pas atteinte.

A titre d'illustration, le protocole naïf (simple et non sûr) d'échange entre le XT et le XμP s'écrit comme mentionné en Figure 2 (en se rappelant qu'exécuter INS_i met à jour i).

Ainsi qu'il apparaît clairement, ce procédé simple est sujet à de nombreuses attaques. Typiquement, un attaquant peut retrouver la valeur d'une clé secrète stockée dans la mémoire du XμP, avec l'aide du programme XJVML suivant:

```
1       getstatic 1
2       store IO
3       getstatic 2
4       store IO
5       getstatic 3
6       store IO
:
:
:
```

Un attaquant pourrait également, par exemple, modifier le montant d'un porte-monnaie électronique en sa faveur.

Nous proposons donc plusieurs modes de réalisation de l'authentification du programme P qui est transmis au XμP.

De manière générale, l'invention concerne un procédé de sécurisation d'un objet portable électronique **XμP** exécutant un programme P fourni par un autre objet électronique non sûr XT, caractérisé en ce qu'il utilise :

- un protocole à clé secrète;
- une clé secrète éphémère K;
- une fonction de MAC μ_K;
- une fonction de hachage $HASH_1$ définie par une fonction de compression H_1 et une constante IV_1;
- une fonction de hachage $HASH_2$ définie par une fonction de compression H_2 et une constante IV_2;
- un identifiant de programme ID stocké dans l'objet électronique **XμP** et valant un hachage de P.

Dans une première partie de l'invention, ce procédé de sécurisation d'un objet portable électronique est caractérisé en ce que le programme P est fourni sous la forme d'une suite de F instructions, F dénotant ainsi le nombre d'instructions de ce programme P.

Dans cette première partie de l'invention, la valeur de ID, qui correspond au hachage du programme P, se calcule en hachant, une à une, les instructions, dans l'ordre croissant des adresses.

De façon plus précise, la première partie de l'invention est caractérisée en ce que ledit protocole comporte les phases suivantes :

a) une phase d'initialisation durant laquelle le XμP génère une clé éphémère K, puis reçoit du XT l'ensemble du programme P, le nombre d'instructions F et son identifiant ID, calcule le haché h de ce programme P avec la fonction $HASH_1$, en utilisant la fonction de compression H_1 et la constante IV_1, et enfin génère des signatures σ_i à l'aide de la fonction μ_K et de la clé K, signatures σ_i qu'il transmet au XT;

b) une phase d'exécution durant laquelle le XμP vérifie l'égalité entre les valeurs de h et de ID, vérifie également que ID est stocké dans sa mémoire non volatile, puis demande, l'une après l'autre, les instructions de P pour les exécuter, et pour certaines d'entre elles, effectue une sous-phase de vérification qui consiste à demander une signature σ, construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et à vérifier cette signature σ;

c) une phase de réaction qui se déroule dès qu'une signature σ est non valable, et qui consiste pour le XμP à prendre les mesures nécessaires contre le XT frauduleux.

Cette première partie de l'invention se décline alors en plusieurs modes, appelés premier, deuxième et troisième modes de réalisation de l'invention.

Dans le premier mode de réalisation, le procédé de sécurisation d'un objet portable électronique est caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de chaque instruction.

De façon plus précise, ce premier mode de réalisation est caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction et retourne à la sous-phase b-1.

Ainsi, de façon préférentielle, le premier mode de sécurisation d'un objet portable électronique selon l'invention est caractérisé en ce qu'il utilise un protocole à clé secrète comprenant les étapes suivantes :

-2. Le XμP génère une clé aléatoire de session *K*, demande au XT l'identifiant ID du programme, le nombre d'instructions *F* qu'il contient et initialise $h \leftarrow IV_1$

-1. Pour i←1 à *F*

(a) Le XμP demande au XT l'instruction numéro *i*

(b) Le XT envoie l'instruction INS_i au XμP

(c) Le XμP calcule la signature $\sigma_i \leftarrow \mu_K(\mathrm{ID}, i, \mathrm{INS}_i)$ et met à jour $h \leftarrow H_1(h, INS_i)$

(d) Le XμP envoie σ_i au XT (aucune copie de σ_i n'est gardée dans le XμP)

(e) Le XT enregistre σ_i

0. Le XμP vérifie que $h = \mathrm{ID}$, que ID est présent en mémoire non volatile (en cas d'échec aller à l'étape 7) et initialise $i \leftarrow 1$

1. Le XμP initialise $v \leftarrow IV_2$

2. Le XT initialise $\sigma \leftarrow IV_2$
3. Le XμP demande au XT l'instruction numéro i
4. Le XT
   (a) met à jour $\sigma \leftarrow H_2(\sigma,\sigma_i)$
   (b) envoie INS_i au XμP
5. Le XμP met à jour $v \leftarrow H_2(v,\mu_K(ID,i,INS_i))$
6. Le XμP
   (a) demande σ au XT et vérifie que $\sigma = v$; en cas d'échec, aller à l'étape 7
   (b) exécute INS_i
   (c) retourne à l'étape 1
7. Le XμP sait que le programme fourni est un programme non authentique, et prend donc toutes les mesures nécessaires défensives de protection.

Dans le précédent paragraphe, IV_1 et IV_2 désignent les vecteurs initiaux des fonctions de hachage $HASH_1$ et $HASH_2$; i est toujours la valeur représentant le compteur de programme; σ_i désigne la signature de l'instruction INS_i. On rappelle que l'exécution de INS_i modifie la valeur de i. Les lettres h, v et σ désignent des variables du protocole dont l'utilisation est expliquée dans ce qui suit.

Le protocole ci-dessus comprend différentes étapes. Nous avons noté par (-2) et (-1) les étapes dites négatives qui se déroulent avant l'exécution du programme P et par (0) à (7) les étapes dites positives qui se déroulent durant l'exécution du programme P.

En étape (-2), le XμP génère de façon aléatoire une clé K éphémère. Cette génération aléatoire peut se faire à l'aide d'un générateur de nombre aléatoire (« random number generator » en anglais) matériel ou à l'aide d'un autre

moyen. De plus, la valeur h est initialisée à la valeur initiale IV_1.

L'étape (-1) est une boucle sur les adresses de programme i. Elle est constituée de sous étapes.

- Dans la sous étape (-1.a), le XμP demande à XT l'instruction d'adresse i
- Dans la sous étape (-1.b), le XT envoie au XμP l'instruction demandée
- Dans la sous étape (-1.c), le XμP calcule la signature symétrique (aussi appelée signature ou MAC) σ_i de l'instruction. De plus, le XμP accumule le hachage du programme dans la valeur h au moyen de la fonction de compression H_1.
- Dans la sous étape (-1.d), le MAC σ_i est envoyé par le XμP au XT.
- Enfin, dans la sous étape (-1.e), le MAC σ_i reçu du XμP est stocké par le XT.

Se déroulent par la suite les étapes ayant lieu pendant l'exécution du programme P.

A l'étape (0), le XμP vérifie que la valeur finale de h (calculée durant la boucle de l'étape (-1)) est égale à la valeur ID, stockée dans sa mémoire non volatile. Grâce à la propriété de non-collision de la fonction de hachage, le XμP est ainsi sûr que le programme pour lequel il a calculé la séquence des MACs σ_i pendant les étapes négatives est effectivement autorisé à l'exécution. De plus, pendant l'étape (0), le compteur de programme i est initialisé à 1. Si la valeur de h diffère de celle de ID, le programme envoyé n'est pas authentique, et la section (7) est exécutée : le XμP prend alors les mesures adéquates contre

l'agression supposée (par exemple, le XμP efface sa mémoire).

Les étapes (1), (2), (3), (4), (5), (6) sont alors répétées un certain nombre de fois, jusqu'à ce que l'instruction finale soit exécutée. Ce procédé de boucle est expliqué dans ce qui suit.

En étape (1), le XμP initialise la variable v à IV_2.

En étape (2), le XT initialise la variable σ à IV_2.

A l'étape (3), le XμP demande au XT l'instruction d'adresse i.

A l'étape (4), le XT remet à jour la variable σ et envoie au XμP l'instruction demandée.

A l'étape (5), le XμP remet à jour la variable v.

L'étape (6) est l'étape critique pour la sécurité. Les sous-étapes (6.a), (6.b) et (6.c) sont effectuées. La sous-étape (6.a) est une sous-étape durant laquelle le XμP demande au XT de lui envoyer la signature collective σ. Le XμP fait alors la comparaison avec la valeur v qu'il a calculée auparavant. Si ces valeurs diffèrent, le programme P reçu n'est pas authentique et l'étape (7) est alors exécutée : le XμP prend alors les mesures appropriées contre cette agression. Si ces valeurs sont égales, le XμP continue l'exécution du protocole en exécutant l'instruction reçue et en retournant à l'étape (1).

Ainsi, dans les étapes négatives, le XμP signe lui-même le programme qui lui est envoyé avec l'aide d'une clé éphémère K, tout en vérifiant que celui ci est correct en comparant le hachage du programme qui lui est envoyé à l'identifiant qu'il contient dans sa mémoire (ID). Dans les étapes positives, il ne reste alors plus qu'à comparer,

pour chaque instruction, la signature fournie par le XT à celle que le XμP recalcule.

Il est ainsi impossible pour le XT d'envoyer une instruction étrangère: il n'a pu faire signer à l'étape (-1) un programme autre que celui d'identifiant ID sans être détecté à l'étape (0), du fait de la propriété de non-collision de la fonction de hachage. Par suite, durant l'exécution des étapes positives, le XT ne peut qu'envoyer des instructions signées par le XμP au cours de l'exécution des étapes négatives, c'est à dire les instructions correspondant effectivement au programme; dans le cas contraire, si le XT essaie d'envoyer des instructions différentes, il ne pourra envoyer la signature correcte lors de la vérification car il ne peut calculer par lui-même les signatures d'autres instructions du fait qu'il ne connaît pas la clé de signature K.

Cette solution est sûre, mais peut être sujette à améliorations.

Le premier mode fait l'objet d'une amélioration qui est un deuxième mode de réalisation de vérification dynamique du programme P qui est envoyé au XμP. Dans ce deuxième mode de réalisation, seules certaines instructions déclenchent une vérification de la signature collective σ.

Pour cela, nous répertorions dans une liste les instructions qui émettent vers l'extérieur du XμP de l'information relative aux données utilisées lors de leur exécution dans le XμP (par exemple les instructions du commandement du port d'entrée-sortie). Ensuite, on ajoute à cette liste d'instructions les instructions qui susceptibles de modifier l'état de la mémoire non volatile du dispositif. Toutes ces instructions sont appelées *critiques pour la sécurité* dans les sections suivantes et

l'ensemble des instructions critiques pour la sécurité est noté S.

Reprenant l'exemple illustratif du langage élémentaire XJVML, nous répertorions ainsi les instructions qui ont, pour certaines valeurs de leurs entrées, un comportement spécial, reconnaissable de l'extérieur. Une instruction est alors appelée "traçable" si la valeur des données utilisées par l'instruction peut influencer la valeur d'une variable physiquement observable (par exemple le compteur de programme). Les instructions 'if L' et 'div' sont donc traçables en raison de leur influence sur le compteur de programme (l'instruction 'div' pouvant provoquer une interruption en cas de nullité du dénominateur). L'inverse de cette notion est celui d' "indistinguabilité en données" qui caractérise les instructions pour lesquelles les données utilisées n'ont aucune influence sur les variables environnementales. Par exemple, l'exécution de l'instruction 'xor' ne révèle pas d'information sur les deux éléments du haut de la pile qui pourrait être observée de l'extérieur du XμP.

Comme l'exécution d'instructions traçables peut révéler de l'information sur des valeurs internes du programme, ces instructions sont par définition critiques pour la sécurité et nous les incluons donc dans S. Par exemple, dans notre jeu d'instruction illustratif XJVML, seules les instructions 'if L' et 'div' sont traçables et l'ensemble S est donc défini comme ci-dessous:

S = {putstatic x, store IO, if L, div}

L'instruction 'store IO' est dans S car elle pourrait déclencher l'émission d'un signal électrique à l'extérieur (par le port d'entrée-sortie). L'instruction 'putstatic x'

est également dans S car elle peut effectuer une modification de la mémoire non volatile.

Ainsi, pour un jeu d'instructions donné, la classification des instructions permettant de définir S nous conduit donc au deuxième mode de réalisation de l'invention tel que décrit dans la section suivante.

Dans ce deuxième mode de réalisation de l'invention, le procédé de sécurisation d'un objet portable électronique est caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction, si celle-ci est une instruction critique pour la sécurité.

De façon plus précise, dans ce second mode, le procédé de sécurisation d'un objet portable électronique est caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) si cette instruction est critique pour la sécurité, alors le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction et retourne à la sous-phase b-1.

De façon préférentielle, toujours dans ce second mode, le procédé de sécurisation d'un objet portable électronique est caractérisé en ce qu'il utilise un ensemble d'instructions critiques pour la sécurité S et en ce que le protocole comprend les étapes suivantes:

-2. Le $X\mu P$ génère une clé aléatoire de session K, demande au XT l'identifiant ID du programme, le nombre d'instructions F qu'il contient et initialise $h \leftarrow IV_1$

-1. Pour i←1 à F

(a) Le $X\mu P$ demande au XT l'instruction numéro i

(b) Le XT envoie l'instruction INS_i au $X\mu P$

(c) Le $X\mu P$ calcule la signature $\sigma_i \leftarrow \mu_K(ID, i, INS_i)$ et met à jour $h \leftarrow H_1(h, INS_i)$

(d) Le $X\mu P$ envoie σ_i au XT (aucune copie de σ_i n'est gardée dans le $X\mu P$)

(e) Le XT enregistre σ_i

0. Le $X\mu P$ vérifie que $h = ID$, que ID est présent en mémoire non volatile (en cas d'échec aller à l'étape 8) et initialise $i \leftarrow 1$

1. Le $X\mu P$ initialise $v \leftarrow IV_2$

2. Le XT initialise $\sigma \leftarrow IV_2$

3. Le $X\mu P$ demande au XT l'instruction numéro i

4. Le XT

(a) met à jour $\sigma \leftarrow H_2(\sigma, \sigma_i)$

(b) envoie INS_i au $X\mu P$

5. Le $X\mu P$ met à jour $v \leftarrow H_2(v, \mu_K(ID, i, INS_i))$

6. Si $INS_i \in S$, le $X\mu P$

(a) demande σ au XT et vérifie que $\sigma = v$; en cas d'échec, aller à l'étape 8

(b) exécute INS_i

(c) retourne à l'étape 1

7. Sinon, le $X\mu P$

(a) exécute INS_i

(b) retourne à l'étape 3

8. Le $X\mu P$ sait que le programme fourni est un programme non

authentique, et prend donc toutes les mesures nécessaires défensives de protection

Dans le précédent paragraphe, IV_1 et IV_2 désignent les vecteurs initiaux des fonctions de hachage $HASH_1$ et $HASH_2$; i dénote toujours la valeur représentant le compteur de programme; σ_i désigne la signature de l'instruction INS_i. On rappelle que l'exécution de INS_i modifie la valeur de *i*. Les lettres h, v et σ désignent des variables du protocole dont l'utilisation est expliquée dans ce qui suit.

Le protocole se compose de différentes étapes. Nous avons noté par (-2) et (-1) les étapes dites négatives qui se déroulent avant l'exécution du programme P et par (0) à (8) les étapes dites positives qui se déroulent durant l'exécution du programme P.

En étape (-2), le XμP génère de façon aléatoire une clé K éphémère. Cette génération aléatoire peut se faire à l'aide d'un générateur de nombre aléatoire (« random number generator » en anglais) matériel ou à l'aide d'un autre moyen. De plus, la valeur h est initialisée à la valeur initiale IV.

L'étape (-1) est une boucle sur les adresses de programme i. Elle est constituée de sous étapes.

- Dans la sous étape (-1.a), le XμP demande à XT l'instruction d'adresse i
- Dans la sous étape (-1.b), le XT envoie au XμP l'instruction demandée
- Dans la sous étape (-1.c), le XμP calcule la signature symétrique (aussi appelée signature ou MAC) σ_i de l'instruction. De plus, le XμP

accumule le hachage du programme dans la valeur h au moyen de la fonction de compression H_1.

- Dans la sous étape (-1.d), le MAC σ_i est envoyé par le XμP au XT.
- Enfin, dans la sous étape (-1.e), le MAC σ_i reçu du XμP est stocké par le XT.

Se déroulent par la suite les étapes ayant lieu pendant l'exécution du programme P.

A l'étape (0), le XμP vérifie que la valeur finale de h calculée durant la boucle de l'étape (-1) est égale à la l'identifiant ID, stocké dans sa mémoire non volatile. Grâce à la propriété de non-collision de la fonction de hachage, le XμP est ainsi certain que le programme pour lequel il a calculé la séquence des MACs σ_i pendant les étapes négatives est effectivement autorisé à l'exécution. De plus, pendant l'étape (0), le compteur de programme i est initialisé à 1. Si la valeur de h diffère de celle de ID, le programme envoyé n'est pas authentique, et la section (8) est exécutée : le XμP prend alors les mesures adéquates contre l'agression supposée (par exemple, le XμP efface sa mémoire).

Les étapes (1), (2), (3), (4), (5), (6), (7) sont alors répétées un certain nombre de fois, jusqu'à ce que l'instruction finale soit exécutée. Ce procédé de boucle est expliqué dans ce qui suit.

En étape (1), le XμP initialise la variable v à IV_2.

En étape (2), le XT initialise la variable σ à IV_2.

A l'étape (3), le XμP demande au XT l'instruction d'adresse i.

A l'étape (4), le XT remet à jour la variable σ et envoie au XμP l'instruction demandée.

A l'étape (5), le XμP remet à jour la variable v.

L'étape (6) est l'étape critique pour la sécurité. Celle ci commence d'abord par un test.

- Si l'instruction reçue INS_i est dans l'ensemble des instructions critiques pour la sécurité S, les sous-étapes (6.a), (6.b) et (6.c) sont effectuées. La sous-étape (6.a) est une sous-étape durant laquelle le XμP demande au XT de lui envoyer la signature collective σ. Le XμP effectue alors la comparaison avec la valeur v qu'il a calculée auparavant. Si ces valeurs sont différentes, le programme P reçu est un programme non-authentique et l'étape (8) est alors exécutée : le XμP prend alors les mesures adéquates contre cette agression (par exemple, le XμP ré-initialise sa mémoire). Si ces valeurs sont égales, le XμP continue l'exécution du protocole en exécutant l'instruction reçue et en retournant à l'étape (1).
- Si l'instruction reçue INS_i n'est pas dans l'ensemble des instructions critiques pour la sécurité S, l'étape (7) est exécutée : le XμP exécute simplement INS_i et continue d'exécuter le procédé en retournant à l'étape (3).

Ainsi, dans les étapes négatives, le XμP signe lui-même le programme qui lui est envoyé (encore une fois avec une clé éphémère), tout en vérifiant que celui ci est authentique en comparant le hachage du programme qui lui est envoyé à l'identifiant de programme qu'il contient dans sa mémoire (ID). Dans les étapes positives, le procédé

permet de vérifier collectivement, aux moments opportuns (c'est à dire pour toutes les instructions critiques pour la sécurité, répertoriées dans l'ensemble S), que les signatures fournies par le XT sont identiques à celles que le XμP avait calculé dans les étapes négatives.

A l'instar du premier mode de réalisation, il est impossible pour le XT d'envoyer au XμP une instruction étrangère au programme: il n'a pu faire signer à l'étape (-1) un programme différent de celui d'identifiant ID sans être détecté à l'étape (0), du fait de la propriété de non-collision de la fonction de hachage. En conséquence, durant l'exécution des étapes positives, le XT ne peut qu'envoyer des instructions signées par le XμP au cours de l'exécution des étapes négatives, c'est à dire les instructions correspondant effectivement au programme; dans le cas contraire, si le XT essaie d'envoyer des instructions différentes, il ne pourra envoyer la signature correcte lors de la vérification car il ne peut calculer par lui-même les signatures d'autres instructions du fait qu'il ne connaît pas la clé de signature K.

Il est cependant encore possible d'améliorer les performances de l'invention à l'aide d'un troisième mode de réalisation de l'invention.

Dans ce troisième mode de réalisation de l'invention, un *niveau de sécurité* est associé à chacune des données manipulées par le XμP. Il permet de distinguer une donnée secrète (par exemple une clé cryptographique stockée en mémoire non volatile) d'une donnée publique (connue ou pouvant être recalculée à partir de données connues). Par concision, nous dénotons par Φ l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné. Il existe plusieurs façon de définir un

niveau de sécurité sur une donnée de calcul, mais l'on peut supposer en toute généralité que l'ensemble Φ des niveaux de sécurité est initialisé à certaines valeurs spécifiques avant l'exécution du programme P, et que le fait d'exécuter une instruction de P peut modifier Φ selon des règles arbitrairement choisies par le concepteur du dispositif.

A titre d'exemple illustratif non limitatif, nous décrivons ci-après une réalisation particulière de ce procédé appliqué à l'architecture XJVML définie plus haut.

Le niveau de sécurité est mis en œuvre sous la forme d'un bit d'information φ selon la convention que sa valeur vaut zéro lorsque la donnée concernée est publique et un lorsque celle-ci est secrète. Plus spécifiquement, la mise en œuvre du procédé concerne les cellules mémoires volatiles (RAM), non volatiles (NVM) et les cellules de pile (ST). Ainsi, on dénote par φ(RAM[j]) le bit de sécurité associé au mot mémoire RAM[j], par φ(NVM[j]) le bit de sécurité associé à NVM[j] et par φ(ST[j]) le bit de sécurité associé à ST[j]. Par convention, les bits de sécurité des cellules NVM sont non volatiles et positionnés à 0 ou 1 par le fabricant du XµP à l'étape de production ou de personnalisation, suivant la nature des données non volatiles correspondantes. Ceux de la mémoire RAM sont initialisées à 0 lors du reset du dispositif. Par convention, φ(IO) est laissé constant à 0 et φ(RNG) est laissé constant à 1. Enfin, les bits de sécurité des éléments de pile désaffectés sont automatiquement remis à 0.

Nous présentons aussi deux règles élémentaires par lesquelles le bit de sécurité d'une nouvelle variable de programme, c'est-à-dire d'une donnée issue d'un calcul à

partir de données précédentes, est établi en fonction de celui desdites données précédentes.

La première règle est que toutes les instructions de transfert ('load', 'getstatic', 'store' et 'putstatic') transfèrent également le bit de sécurité de la variable transférée. La seconde règle est appliquée aux instructions arithmétiques et logiques. Elle définit chaque bit de sécurité des variables de sortie de l'instruction concernée comme le OU logique des bits de sécurité de toutes les variables d'entrée de l'instruction. Autrement dit, aussitôt qu'une donnée secrète entre dans le calcul, toutes les données qui en découlent sont répertoriées comme étant secrètes. Cette règle peut notamment mais non uniquement être facilement câblée en hardware comme un simple OU booléen ("OR", dénoté V dans la Figure 5) pour les instructions binaires (c'est à dire avec deux arguments d'entrée). Par souci de clarté, nous fournissons dans la Figure 5 la sémantique dynamique des instructions de XJVML sur Φ.

Etant donné maintenant un jeu d'instruction quelconque, et les règles permettant de définir au cours du temps l'ensemble des niveaux de sécurité Φ des données utilisées par l'exécution d'un programme, nous y associons le procédé de l'invention tel que décrit par son second mode de réalisation. Le principe du troisième mode de réalisation repose sur le fait que la vérification collective des instructions émises par le XT, jusqu'alors déclenchée par la détection d'une instruction critique pour la sécurité, peut être épargnée dès lors que cette instruction n'utilise par exemple que des données répertoriées comme publiques. Une vérification de MAC n'est effectivement pas nécessairement invoquée dans ce cas puisque le danger inhérent à l'exécution d'une instruction

critique se trouve annulé par le fait que celle-ci ne peut fournir des informations que sur des données préalablement connues ou modifier de telles données.

Par concision, on dénote par Alert(INS,Φ) la fonction booléenne (c'est-à-dire retournant VRAI ou FAUX) qui détermine si l'exécution de l'instruction critique INS occasionne ou non une vérification lorsque le niveau de sécurité des données d'entrée que cette instruction manipule est donné par Φ.

Dans notre exemple de mise en œuvre dans le cadre du langage XJVML, la fonction Alert peut être définie de plusieurs manières différentes ainsi qu'illustré sur les Figures 3 et 4.

Ainsi, nous définissons un troisième mode de réalisation de l'invention, caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction si celle-ci est une instruction critique pour la sécurité, et si l'une au moins des données utilisées par cette instruction est une donnée secrète.

De façon plus précise, dans ce troisième mode, ce procédé de sécurisation d'un objet portable électronique est caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné P et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) si cette instruction est critique pour la sécurité et si l'une au moins des données utilisées par l'instruction est secrète, alors le XμP demande

une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction, met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution, et retourne à la sous-phase b-1.

Décliné avec l'utilisation de la fonction booléenne Alert, ce troisième mode de réalisation est caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné P, et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) si cette instruction est critique pour la sécurité et si la fonction booléenne Alert déterminée à partir du niveau de sécurité des données utilisées par l'instruction et par la nature de l'instruction elle-même s'évalue en VRAI, alors le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction, met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution, et retourne à la sous-phase b-1.

De façon préférentielle, ce troisième mode de réalisation est caractérisé en ce qu'il utilise un ensemble d'instructions critiques pour la sécurité S et en ce que le protocole comprend les étapes suivantes:

-2. Le $\mathbf{X\mu P}$ génère une clé aléatoire de session K, demande au XT l'identifiant ID du programme, le nombre d'instructions F qu'il contient et initialise $h \leftarrow IV_1$

-1. Pour i←1 à F

(a) Le $\mathbf{X\mu P}$ demande au XT l'instruction numéro i

(b) Le XT envoie l'instruction $\mathbf{INS}_i$ au $\mathbf{X\mu P}$

(c) Le $\mathbf{X\mu P}$ calcule la signature $\sigma_i \leftarrow \mu_K(\mathbf{ID}, i, \mathbf{INS}_i)$ et met à jour $h \leftarrow H_1(h, INS_i)$

(d) Le $\mathbf{X\mu P}$ envoie σ_i au XT (aucune copie de σ_i n'est gardée dans le $\mathbf{X\mu P}$)

(e) Le XT enregistre σ_i

0. Le $\mathbf{X\mu P}$ vérifie que $h = \mathbf{ID}$, que ID est présent en mémoire non volatile (en cas d'échec aller à l'étape 8) et initialise $i \leftarrow 1$

1. Le $\mathbf{X\mu P}$ initialise $\nu \leftarrow IV_2$

2. Le XT initialise $\sigma \leftarrow IV_2$

3. Le $\mathbf{X\mu P}$ demande au XT l'instruction numéro i

4. Le XT

(a) met à jour $\sigma \leftarrow H_2(\sigma, \sigma_i)$

(b) envoie $\mathbf{INS}_i$ au $\mathbf{X\mu P}$

5. Le $\mathbf{X\mu P}$ met à jour $\nu \leftarrow H_2(\nu, \mu_K(ID, i, INS_i))$

6. Si $INS_i \in S$ et Alert(INS_i,Φ)=VRAI, le $\mathbf{X\mu P}$

(a) demande σ au XT et vérifie que $\sigma = \nu$; en cas d'échec, aller à l'étape 8

(b) exécute $\mathbf{INS}_i$

(c) met à jour Φ

(d) retourne à l'étape 1

7. Sinon, le XμP

(a) exécute INS_i

(b) met à jour Φ

(c) retourne à l'étape 3

8. Le XμP sait que le programme fourni est un programme non authentique, et prend donc toutes les mesures nécessaires défensives de protection.

Ainsi, par différence avec le protocole décrit dans le second mode de réalisation de l'invention, une vérification de la signature collective à l'étape 6 n'est effectuée que lorsque la fonction Alert s'évalue en VRAI juste avant que l'instruction critique soit exécutée.

En fonction de la mise en œuvre de ladite fonction, le concepteur de l'architecture obtient ainsi un moyen de vérifier le programme en fonction du contexte, c'est-à-dire en évitant dans le protocole le déclenchement d'une vérification considérée comme inutile au regard du niveau de sécurité des données en jeu.

Dans une deuxième partie de l'invention, le programme est authentifié par groupe d'instructions, et non plus par instructions simples. Les instructions peuvent en effet être regroupées sous la forme de petits blocs appelés sections qui permettent de limiter le nombre de signatures générées et vérifiées par le XμP.

Suivant la définition classique des documents « Advanced Compiler Design and Implementation », de S. Muchnick, publié en 1997 et « Compilers: Principles, Techniques, and Tools », de A. Aho, R. Sethi et J. Ullman, publié en 1986, nous appelons « bloc de base » une suite séquentielle et ordonnée d'instructions, que l'on ne peut exécuter qu'en exécutant la première et la dernière

instruction. L'homme de métier décrit habituellement l'ensemble des blocs de base d'un programme P sous la forme d'un graphe CFG(P) (CFG signifiant « control flow graph » en anglais), calculé par des moyens connus d'analyse de flot de contrôle (expliqués notamment dans les documents « Identifying Loops in Almost Linear Time », de G. Ramalingam, publié en 1999, et « Advanced Compiler Design and Implementation », de S. Muchnick, publié en 1997). Dans un tel graphe, les nœuds sont identifiés aux blocs de base et les arêtes symbolisent les dépendances de flot de contrôle (« control flow dependancies »).

La présence d'une arête B_0 -> B_1 dans le graphe (on dit alors que B_1 est un fils de B_0 et B_0 un père de B_1) signifie que la dernière instruction du bloc B_0 peut transférer le contrôle du programme à la première instruction de B_1.

Lorsque B_0 -> B_1, B_0 => B_1 signifie que B_0 n'a aucun autre fils que B_1 (mais B_1 peut avoir d'autres pères que B_0). Nous définissons maintenant une notion légèrement différente de celle des blocs de base, que nous appelons *section de programme*.

De manière rigoureuse, une section de programme est une suite maximale de blocs de base B_0 => B_1 => B_2 => ... => B_z telle que ni l'instruction de fin de programme ('halt' en XJVML) ni aucune instruction de S (instruction critique) n'apparaisse dans les blocs sauf éventuellement comme dernière instruction de B_z. La section est alors dénotée par $s = <B_0, B_1, ..., B_z>$. Dans une section de programme, comme dans les blocs de base, le flot de contrôle est déterministe, c'est à dire indépendant de la valeur que les variables de programme sont susceptibles de prendre pendant l'exécution.

Il est connu que le calcul des blocs de base d'un programme peut être fait dans un temps presque linéaire en le nombre d'instruction de ce programme (« Identifying Loops in Almost Linear Time », de G. Ramalingam, publié en 1999) et l'homme de l'art verra facilement que les algorithmes permettant de calculer CFG(P) à partir de P peuvent être modifiés de façon simple pour calculer, de manière également performante, l'ensemble des sections du programme P. Ainsi, les sections de P peuvent être calculées facilement lors la compilation de P.

La deuxième partie de l'invention se décline en un quatrième, cinquième et sixième modes de réalisation de l'invention que nous décrivons maintenant. Dans ces modes, les signatures symétriques générées par le XµP authentifient des sections plutôt que des instructions individuelles du programme.

Au contraire des trois premiers modes de réalisation de la première partie de l'invention, dans lesquels le programme était fourni sous forme de suite d'instructions, ces quatrième, cinquième et sixième modes de réalisation de l'invention sont des procédés de sécurisation d'un objet portable électronique caractérisés en ce que le programme P est fourni sous la forme d'une suite de sections ou blocs d'instructions, G dénotant le nombre de sections de ce programme P, et en ce qu'il utilise une troisième fonction de hachage, nommée $HASH_3$, définie par une fonction de compression H_3 et une constante IV_3.

Dans cette seconde partie de l'invention, la valeur de ID, qui correspond au hachage du programme P, se calcule en hachant, une à une, les sections, dans l'ordre croissant des adresses de ces sections, puis finalement en hachant les hachés des sections dans l'ordre croissant des adresses de départ des sections.

De façon plus précise, la deuxième partie de l'invention est caractérisée en ce que ledit protocole comporte les phases suivantes :

a) une phase d'initialisation durant laquelle le XμP génère une clé éphémère K, puis reçoit du XT l'ensemble du programme P, son nombre de sections G et son identifiant ID, calcule le haché h de ce programme P à l'aide de la fonction $HASH_1$, en utilisant la fonction de compression H_1 et la constante IV_1, et à l'aide de la fonction $HASH_3$, en utilisant la fonction de compression H_3 et la constante IV_3, et enfin génère des signatures σ_j à l'aide de la fonction μ_K et de la clé K, signatures σ_j qu'il transmet au XT;

b) une phase d'exécution durant laquelle le XμP vérifie l'égalité entre les valeurs de h et de ID, vérifie également que ID est stocké dans sa mémoire non volatile, puis demande, l'une après l'autre, les sections de P pour les exécuter, effectue ensuite une sous-phase de vérification de conformité de ces sections, puis finalement, pour l'instruction finale de certaines sections, effectue une sous-phase de vérification qui consiste à demander une signature σ, construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et à la vérifier;

c) une phase de réaction qui se déroule dès qu'une signature σ est non valable ou qu'une section est non conforme, et qui consiste pour le XμP à prendre les mesures nécessaires contre le XT frauduleux.

Plus précisément, la sous-phase de vérification de conformité d'une section donnée consiste à vérifier qu'aucune instruction de cette section, sauf éventuellement la dernière, n'est une instruction critique pour la sécurité.

Cette deuxième partie de l'invention se décline alors en plusieurs modes, appelés quatrième, cinquième et sixième mode de réalisation de l'invention.

Le quatrième mode de réalisation se caractérise en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction finale de chaque section.

De façon plus précise, ce quatrième mode de réalisation est caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une section au XT;

b-2) pour chaque instruction non finale de la section demandée, le XμP vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-4) le XμP retourne alors à la sous-phase b-1.

De façon préférentielle, le quatrième mode de réalisation de l'invention est caractérisé en ce qu'il utilise un protocole à clé secrète comprenant les étapes suivantes :

-2. Le XμP génère une clé aléatoire de session *K*, demande au XT l'identifiant ID du programme, le nombre de sections G qu'il contient et initialise $h \leftarrow IV_1$

-1. Pour *j*←1 à *G*

(a) Le XμP demande au XT la section numéro *j*, le nombre *t* d'instructions dans cette section et initialise $g \leftarrow IV_3$

(b) Pour *i*←1 à *t*, le XT envoie l'instruction INS_i au XμP qui met à jour $g \leftarrow H_3(g, INS_i)$

(c) Le XμP calcule la signature $\sigma_j \leftarrow \mu_K(ID, j, g)$ de la section et met à jour $h \leftarrow H_1(h, g)$

(d) Le XμP envoie σ_j au XT (aucune copie de σ_j n'est gardée dans le XμP)

(e) Le XT enregistre σ_j

0. Le XμP vérifie que $h = ID$, que ID est présent en mémoire non volatile (en cas d'échec aller à l'étape 9) et initialise $j \leftarrow 1$

1. Le XμP initialise $\nu \leftarrow IV_2$

2. Le XT initialise $\sigma \leftarrow IV_2$

3. Le XμP demande au XT la section numéro *j*, le nombre *t* d'instructions qui la compose et initialise $g \leftarrow IV_3$ et $i \leftarrow 1$

4. Le XT met à jour $\sigma \leftarrow H_2(\sigma, \sigma_j)$ et initialise $i \leftarrow 1$

5. Le XT envoie INS_i au XμP et incrémente $i \leftarrow i+1$

6. Le XμP met à jour $g \leftarrow H_3(g, INS_i)$

7. Si i<t, alors le XμP

(a) teste si $INS_t \in S$, et dans ce cas, aller à l'étape 9

(b) exécute INS_t

(c) retourne à l'étape 5

8. Si i=t, alors le XμP

(a) met à jour $v \leftarrow H_2(v, \mu_K(ID, j, g))$

(b) demande σ au XT et vérifie que $\sigma = v$; en cas d'échec, aller à l'étape 9

(c) exécute INS_t

(d) retourne à l'étape 1

9. Le XμP sait que le programme fourni est un programme non authentique, et prend donc toutes les mesures nécessaires défensives de protection.

Dans le précédent paragraphe, et dans la suite (pour les cinquièmes et sixième mode de réalisation), la signature d'une section S_j dont la première instruction a pour adresse j et constituée des instructions INS_1, …, INS_k peut être définie à titre d'exemple par :

$\sigma_j = \mu(ID, j, g)$ où g désigne $g = HASH_3(INS_1, \ldots, INS_k)$

$HASH_3$ étant ici une fonction de hachage définie par une fonction de compression H_3 et un vecteur d'initialisation IV_3 conformément à l'état de l'art. Le fait d'employer la définition classique du hachage par itération est indispensable à notre quatrième, cinquième et sixième mode de réalisation.

Le quatrième mode de réalisation se compose lui aussi d'étapes négatives et positives. Nous expliquons brièvement son fonctionnement, celui ci étant très proche des premiers modes de réalisation. Dans l'étape (-2), une clé aléatoire K est générée, l'identifiant ID et le nombre de sections G sont demandés. Puis h est initialisé à IV_1. Dans l'étape (-

1), le programme P est signé à l'aide de la clé K et de la fonction de MAC μK. Ici, les signatures sont des signatures par section. Les signatures σ_j sont générées par le XμP et envoyées ensuite au XT, qui les stocke. Dans l'étape (0), le XμP vérifie que le programme est correct, en vérifiant que le haché calculé est identique à ID, et que ID est présent dans sa mémoire non volatile. Les étapes (1) et (2) sont des étapes d'initialisation pour le XμP et le XT. En étape (3), le XμP demande au XT le nombre d'instruction t de la section courante, et initialise g à IV_3. Le XT remet quant à lui à jour la variable σ en étape (4) et initialise i à 1. En étape (5), l'instruction courante de la section courante est envoyée au XμP et i est incrémenté. Le XμP remet alors à jour g, variable qui lui sert à accumuler le hachage de la section courante. L'étape (6) est celle de la vérification de conformité de la section : le XμP y vérifie que toutes les instructions non finales sont non critiques. Il exécute également ces instructions. L'étape (7) est celle qui se déroule pour l'instruction finale de la section : le XμP demande alors une signature et en vérifie l'authenticité. En cas de succès, l'instruction est exécutée, et le procédé repart de l'étape 1. Enfin, à tout moment, si une section est non conforme, ou si une signature est fausse, l'étape (9), qui est celle de l'étape de réaction, est exécutée : le XμP prend alors les mesures nécessaires de protection.

A la différence des modes de réalisation précédents, chaque section ne peut occasionner au plus qu'une vérification de MAC. On se rappellera en effet qu'une instruction critique pour la sécurité ne peut se trouver

qu'en tant que dernière instruction d'une section. Par définition, la dernière instruction INS_k d'une section est:

- soit une instruction de S. Dans ce cas, son exécution peut déclencher ou non une vérification de signature, selon la politique de sécurité Alert(INS,Φ)
- soit l'instruction de fin de programme ('halt' en XJVML), qui interrompt l'exécution.

Reprenant les idées des deuxième et troisième modes de réalisation, mais appliquées à un programme P donné sous la forme de sections, nous dérivons les cinquièmes et sixièmes modes de réalisation de l'invention.

Le cinquième mode de réalisation de l'invention est un procédé de sécurisation d'un objet portable électronique, du type deuxième partie de l'invention (c'est à dire avec un programme P donné sous la forme de sections), caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction finale de chaque section, si ladite instruction est une instruction critique pour la sécurité.

Plus précisément, ce cinquième mode est caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le **XμP** demande une section au XT;

b-2) pour chaque instruction non finale de la section demandée, le **XμP** vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) si l'instruction est critique pour la sécurité, le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-4) le XμP retourne alors à la sous-phase b-1.

De façon préférentielle, ce cinquième mode est caractérisé en ce qu'il utilise un ensemble d'instructions critiques pour la sécurité S et en ce que le protocole comprend les étapes suivantes:

-2. Le XμP génère une clé aléatoire de session *K*, demande au XT l'identifiant ID du programme, le nombre de sections G qu'il contient et initialise $h \leftarrow IV_1$

-1. Pour *j*←1 à *G*

(a) Le XμP demande au XT la section numéro *j*, le nombre *t* d'instructions dans cette section et initialise $g \leftarrow IV_3$

(b) Pour *i*←1 à *t*, le XT envoie l'instruction INS_i au XμP qui met à jour $g \leftarrow H_3(g, INS_i)$

(c) Le XμP calcule la signature $\sigma_j \leftarrow \mu_K(ID, j, g)$ de la section et met à jour $h \leftarrow H_1(h, g)$

(d) Le XμP envoie σ_j au XT (aucune copie de σ_j n'est gardée dans le XμP)

(e) Le XT enregistre σ_j

0. Le XμP vérifie que $h = ID$, que ID est présent en mémoire non volatile (en cas d'échec aller à l'étape 10) et initialise $j \leftarrow 1$

1. Le $X\mu P$ initialise $v \leftarrow IV_2$
2. Le XT initialise $\sigma \leftarrow IV_2$
3. Le $X\mu P$ demande au XT la section numéro j, le nombre t d'instructions qui la compose et initialise $g \leftarrow IV_3$ et $i \leftarrow 1$
4. Le XT met à jour $\sigma \leftarrow H_2(\sigma, \sigma_j)$ et initialise $i \leftarrow 1$
5. Le XT envoie INS_i au $X\mu P$ et incrémente $i \leftarrow i+1$
6. Le $X\mu P$ met à jour $g \leftarrow H_3(g, INS_i)$
7. Si i<t, alors le $X\mu P$
   (a) teste si $INS_i \in S$, et dans ce cas, aller à l'étape 10
   (b) exécute INS_i
   (c) retourne à l'étape 5
8. Si i=t et $INS_i \in S$, alors le $X\mu P$
   (a) met à jour $v \leftarrow H_2(v, \mu_K(ID, j, g))$
   (b) demande σ au XT et vérifie que $\sigma = v$; en cas d'échec, aller à l'étape 10
   (c) exécute INS_i
   (d) retourne à l'étape 1
9. Si i=t et $INS_i \notin S$, alors le $X\mu P$
   (a) met à jour $v \leftarrow H_2(v, \mu_K(ID, j, g))$
   (b) exécute INS_i
   (c) retourne à l'étape 3
10. Le $X\mu P$ sait que le programme fourni est un programme non authentique, et prend donc toutes les mesures nécessaires défensives de protection.

Ce cinquième mode de réalisation de l'invention est très proche du quatrième, et nous n'allons expliquer ici que les phases différentes de celui ci, c'est à dire les phases 8 et 9. Dans le quatrième mode de réalisation, toutes les instructions finales des sections faisaient

l'objet d'une vérification de signature. Ici, en étape (8), on teste l'instruction finale : si elle est critique, on demande une signature. Par contre, si l'instruction finale n'est pas critique, alors, dans l'étape (9), on exécute l'instruction sans demander de signature, et on continue le protocole en retournant à l'étape 3.

Comme on peut le voir, l'avantage est grand : seules certaines instructions finales feront l'objet d'une vérification de signature, et ainsi, le protocole sera d'autant plus rapide.

Mais il est encore possible d'apporter une dernière amélioration au protocole, faisant l'objet du sixième mode de réalisation de l'invention.

Celui ci est un procédé de sécurisation d'un objet portable électronique caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction finale de chaque section, si ladite instruction est une instruction critique pour la sécurité, et si l'une au moins des données utilisées par cette instruction est une donnée secrète.

Plus précisément, le sixième mode de réalisation de l'invention est un procédé de sécurisation d'un objet portable électronique caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une section au XT;

b-2) pour chaque instruction non finale de la section demandée, le XμP vérifie si cette instruction est critique, effectuant dans ce cas la phase de

réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) si l'instruction est critique pour la sécurité et si l'une au moins des données utilisées par l'instruction est secrète, le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-33) le XμP met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution ;

b-4) le XμP retourne alors à la sous-phase b-1.

Une autre façon de réaliser le sixième mode de réalisation de l'invention est d'utiliser un protocole, caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné, en ce qu'il utilise une fonction booléenne Alert et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) pour chaque instruction non finale de la section demandée, le XμP vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) si l'instruction est critique pour la sécurité et si la fonction booléenne Alert déterminée à partir du niveau de sécurité des données utilisées par l'instruction et par la nature de l'instruction elle-même s'évalue en VRAI, le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-33) le XμP met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution ;

b-4) le XμP retourne alors à la sous-phase b-1.

Ainsi, de manière préférentielle, le sixième mode de réalisation de l'invention est caractérisé en ce qu'il utilise un ensemble d'instructions critiques pour la sécurité S, et en ce qu'il comprend les étapes suivantes:

-2. Le XμP génère une clé aléatoire de session *K*, demande au XT l'identifiant ID du programme, le nombre de sections G qu'il contient et initialise $h \leftarrow IV_1$

-1. Pour *j*←1 à *G*

(a) Le XμP demande au XT la section numéro *j*, le nombre *t* d'instructions dans cette section et initialise $g \leftarrow IV_3$

(b) Pour *i*←1 à *t*, le XT envoie l'instruction INS_i au XμP qui met à jour $g \leftarrow H_3(g, INS_i)$

(c) Le XμP calcule la signature $\sigma_j \leftarrow \mu_K(ID, j, g)$ de la

section et met à jour $h \leftarrow H_1(h,g)$

(d) Le $X\mu P$ envoie σ_j au XT (aucune copie de σ_j n'est gardée dans le $X\mu P$)

(e) Le XT enregistre σ_j

0. Le $X\mu P$ vérifie que $h = ID$, que ID est présent en mémoire non volatile (en cas d'échec aller à l'étape 10) et initialise $j \leftarrow 1$

1. Le $X\mu P$ initialise $v \leftarrow IV_2$

2. Le XT initialise $\sigma \leftarrow IV_2$

3. Le $X\mu P$ demande au XT la section numéro j, le nombre t d'instructions qui la compose et initialise $g \leftarrow IV_3$ et $i \leftarrow 1$

4. Le XT met à jour $\sigma \leftarrow H_2(\sigma,\sigma_j)$ et initialise $i \leftarrow 1$

5. Le XT envoie INS_i au $X\mu P$ et incrémente $i \leftarrow i+1$

6. Le $X\mu P$ met à jour $g \leftarrow H_3(g,INS_i)$

7. Si i<t, alors le $X\mu P$

(a) teste si $INS_i \in S$, et dans ce cas, aller à l'étape 10

(b) exécute INS_i

(c) met à jour Φ

(d) retourne à l'étape 5

8. Si i=t et $INS_i \in S$ et Alert(INS_i,Φ)=VRAI, alors le $X\mu P$

(a) met à jour $v \leftarrow H_2(v,\mu_K(ID,j,g))$

(b) demande σ au XT et vérifie que $\sigma = v$; en cas d'échec, aller à l'étape 10

(c) exécute INS_i

(d) met à jour Φ

(e) retourne à l'étape 1

9. Si i=t et ($INS_i \notin S$ ou Alert(INS_i,Φ)=FAUX), alors le $X\mu P$

(a) met à jour $v \leftarrow H_2(v,\mu_K(ID,j,g))$

(b) exécute INS_i

(c) met à jour Φ

(d) retourne à l'étape 3

10. Le XμP sait que le programme fourni est un programme non authentique, et prend donc toutes les mesures nécessaires défensives de protection.

La différence de ce dernier mode de réalisation avec le cinquième mode de réalisation est minime, et est expliquée dans ce qui suit : en étape (8), on ne teste pas seulement si l'instruction finale est critique pour la sécurité, mais aussi si une des données d'entrée de l'instruction est secrète (ceci nous est donné par la condition Alert(INS_t,Φ)=VRAI). Si ces deux conditions sont réunies, une vérification de signature est enclenchée, l'instruction est ensuite exécutée, et le protocole redémarre de l'étape (1). Par contre, dans le cas contraire, l'instruction est exécutée sans déclencher de vérification de signature, et le protocole redémarre de l'étape (3).

Comme pourra le voir l'homme de l'art, le dernier protocole minimise au maximum le nombre de signatures demandées au XT, tout en garantissant la sécurité du XμP.

Dans le deuxième ou troisième modes de la première partie de l'invention, et dans le quatrième, cinquième ou sixième mode de la deuxième partie de l'invention, le procédé est caractérisé en ce qu'au moins un des types suivants d'instructions sont critiques pour la sécurité :

- les instructions de test et/ou
- les instructions émettant de l'information vers l'extérieur par un moyen de communication et/ou
- les instructions modifiant le contenu de la mémoire non-volatile et/ou

- les instructions de calcul présentant des cas particuliers lors de leur exécution, tels que le lancement d'exceptions.

De plus, les troisième et sixième modes sont préférentiellement caractérisés en ce que la fonction booléenne Alert s'évalue en VRAI pour au moins un des types suivants d'instructions:

- les instructions de test et/ou
- les instructions émettant de l'information vers l'extérieur par un moyen de communication et/ou
- les instructions modifiant le contenu de la mémoire non-volatile et/ou
- les instructions de calcul présentant des cas particuliers lors de leur exécution, tels que le lancement d'exceptions.

Dans une solution encore plus efficace, les troisième et sixième modes sont caractérisés en ce que la fonction booléenne Alert s'évalue en VRAI pour au moins un des types suivants d'instructions, si au moins une des données d'entrée est secrète, et FAUX si toutes les données testées sont publiques:

- les instructions de test et/ou
- les instructions émettant de l'information vers l'extérieur par un moyen de communication et/ou
- les instructions modifiant le contenu de la mémoire non-volatile et/ou
- les instructions de calcul présentant des cas particuliers lors de leur exécution, tels que le lancement d'exceptions.

Pour les troisième et sixième modes, l'ensemble des niveaux de sécurité Φ utilisé lors de l'exécution d'un programme P est préférentiellement indiqué par la valeur d'une fonction φ, telle que, pour toute donnée u utilisée par le programme, $\varphi(u)=0$ désigne le fait que u est publique et $\varphi(u)=1$ désigne le fait que u est privée, et telle que, pour toute donnée v résultant de l'exécution d'une instruction du programme P, $\varphi(v)=1$ si au moins une des données d'entrée de l'instruction est privé, et, sinon, $\varphi(v)=0$.

Plus précisément, les valeurs de la fonction φ sont calculées au moyen d'une mise en œuvre matérielle d'une fonction « OU logique » réalisée sur les valeurs de la fonction φ pour les données d'entrée des instructions.

Enfin, par souci de simplicité et de pratique, les fonctions de hachage $HASH_1$, $HASH_2$ et $HASH_3$ peuvent être identiques.

La présente invention s'applique à un objet électronique caractérisé en ce qu'il met en œuvre l'ensemble des modes de réalisation de l'invention tels que décrits ci-dessus.

REVENDICATIONS

1. Procédé de sécurisation d'instructions d'un objet portable électronique **XμP** exécutant un programme P fourni par un autre objet électronique non sûr XT sous la forme d'une suite de F instructions, F dénotant ainsi le nombre d'instructions de ce programme P, procédé utilisant

- un protocole à clé secrète coopérant avec une clé secrète éphémère K;
- une fonction cryptographique symétrique MAC μ_K coopérant avec une fonction de hachage $HASH_1$, définie par une fonction de compression H_1 et une constante IV_1, et une fonction de hachage $HASH_2$ définie par une fonction de compression H_2 et une constante IV_2;
- un identifiant de programme ID stocké dans l'objet électronique **XμP** et valant un hachage de P,

procédé caractérisé en ce que ledit protocole à clé publique comporte les phases suivantes :

a) une phase d'initialisation durant laquelle le **XμP** génère une clé éphémère K, puis reçoit du XT l'ensemble du programme P, le nombre d'instructions F et son identifiant ID, calcule le haché h de ce programme P avec la fonction $HASH_1$, en utilisant la fonction de compression H_1 et la constante IV_1, et enfin génère des signatures σ_i à l'aide de la fonction μ_K et de la clé K, signatures σ_i qu'il transmet au XT;

b) une phase d'exécution durant laquelle le XμP vérifie l'égalité entre les valeurs de h et de ID, vérifie également que ID est stocké dans sa mémoire non volatile, puis demande, l'une après l'autre, les instructions de P pour les exécuter, et pour certaines d'entre elles, effectue une sous-phase de vérification qui consiste à demander une signature σ, construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide de la fonction $HASH_2$, et à vérifier cette signature σ;

c) une phase de réaction qui se déroule dès qu'une signature σ est non valable.

2. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 1 caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de chaque instruction.

3. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 2 caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide la fonction

$HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction et retourne à la sous-phase b-1.

4. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 1 caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction, si celle-ci est une instruction critique pour la sécurité.

5. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 4 caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) si cette instruction est critique pour la sécurité, alors le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction et retourne à la sous-phase b-1.

6. Procédé de sécurisation d'un objet portable électronique selon la revendication 1 caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction si

celle-ci est une instruction critique pour la sécurité, et si l'une au moins des données utilisées par cette instruction est une donnée secrète.

7. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 6 caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné P et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) si cette instruction est critique pour la sécurité et si l'une au moins des données utilisées par l'instruction est secrète, alors le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction, met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution, et retourne à la sous-phase b-1.

8. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 7 caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné P, en ce qu'il utilise une fonction

booléenne Alert et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une instruction au XT;

b-2) si cette instruction est critique pour la sécurité et si la fonction booléenne Alert déterminée à partir du niveau de sécurité des données utilisées par l'instruction et par la nature de l'instruction elle-même s'évalue en VRAI, alors le XμP demande une signature σ construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-3) le XμP exécute l'instruction, met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution, et retourne à la sous-phase b-1.

9. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 1, caractérisé en ce qu'il utilise une fonction de hachage $HASH_3$ définie par une fonction de compression H_3 et une constante IV_3, et en ce que le programme P est fourni sous la forme d'une suite de G sections ou blocs d'instructions, G dénotant ainsi le nombre de sections dudit programme.

10. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 9 caractérisé en ce que ledit protocole comporte les phases suivantes :

a) une phase d'initialisation durant laquelle le XμP génère une clé éphémère K, puis reçoit du XT l'ensemble du programme P, son nombre de sections G et son identifiant ID, calcule le haché h de ce programme P à l'aide de la fonction $HASH_1$, en utilisant la fonction de compression H_1 et la constante IV_1, et à l'aide de la fonction $HASH_3$, en utilisant la fonction de compression H_3 et la constante IV_3, et enfin génère des signatures σ_j à l'aide de la fonction μ_K et de la clé K, signatures σ_j qu'il transmet au XT;

b) une phase d'exécution durant laquelle le XμP vérifie l'égalité entre les valeurs de h et de ID, vérifie également que ID est stocké dans sa mémoire non volatile, puis demande, l'une après l'autre, les sections de P pour les exécuter, effectue ensuite une sous-phase de vérification de conformité de ces sections, puis finalement, pour l'instruction finale de certaines sections, effectue une sous-phase de vérification qui consiste à demander une signature σ, construite à partir des signatures σ_i générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et à la vérifier;

c) une phase de réaction qui se déroule dès qu'une signature σ est non valable ou qu'une section est non conforme.

11. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 10

caractérisé en ce que la sous-phase de vérification de conformité d'une section donnée consiste à vérifier qu'aucune instruction de cette section, sauf éventuellement la dernière, n'est une instruction critique pour la sécurité.

12. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 11 caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction finale de chaque section.

13. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 12 caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une section au XT;

b-2) pour chaque instruction non finale de la section demandée, le XμP vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-4) le XμP retourne alors à la sous-phase b-1.

14. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 11 caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction finale de chaque section, si ladite instruction est une instruction critique pour la sécurité.

15. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 14 caractérisé en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une section au XT;

b-2) pour chaque instruction non finale de la section demandée, le XμP vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) si l'instruction est critique pour la sécurité, le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-4) le XμP retourne alors à la sous-phase b-1.

16. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 11 caractérisé en ce que la sous-phase de vérification dans la phase d'exécution est une vérification de la signature σ se déroulant avant l'exécution de l'instruction finale de chaque section, si ladite instruction est une instruction critique pour la sécurité, et si l'une au moins des données utilisées par cette instruction est une donnée secrète.

17. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 16 caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le XμP demande une section au XT;

b-2) pour chaque instruction non finale de la section demandée, le XμP vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) si l'instruction est critique pour la sécurité et si l'une au moins des données utilisées par l'instruction est secrète, le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la

phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le **XμP** exécute l'instruction;

b-33) le **XμP** met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution ;

b-4) le **XμP** retourne alors à la sous-phase b-1.

18. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 16 caractérisé en ce qu'il utilise une variable Φ définissant l'ensemble des niveaux de sécurité définis à un instant donné par l'exécution d'un programme donné, en ce qu'il utilise une fonction booléenne Alert et en ce que la phase d'exécution comporte les sous-phases suivantes :

b-1) le **XμP** demande une instruction au XT;

b-2) pour chaque instruction non finale de la section demandée, le **XμP** vérifie si cette instruction est critique, effectuant dans ce cas la phase de réaction, et sinon exécute cette instruction et passe à l'instruction suivante;

b-3) pour l'instruction finale de la section demandée :

b-31) si l'instruction est critique pour la sécurité et si la fonction booléenne Alert déterminée à partir du niveau de sécurité des données utilisées par l'instruction et par la nature de l'instruction elle-même s'évalue en

VRAI, le XμP demande une signature σ construite à partir des signatures σ_j générées lors de la phase d'initialisation et à l'aide la fonction $HASH_2$, et, en cas de non-validité de cette signature σ, exécute la phase de réaction;

b-32) le XμP exécute l'instruction;

b-33) le XμP met à jour le niveau de sécurité (donnée secrète ou donnée non secrète) de chacune des données issues de l'exécution ;

b-4) le XμP retourne alors à la sous-phase b-1.

19. Procédé de sécurisation d'instructions d'un objet portable électronique selon l'une quelconque des revendications 4 à 8 ou 11 à 18, caractérisé en ce qu'au moins un des types suivants d'instructions sont critiques pour la sécurité :
    - les instructions de test et/ou
    - les instructions émettant de l'information vers l'extérieur par un moyen de communication et/ou
    - les instructions modifiant le contenu de la mémoire non-volatile et/ou
    - les instructions de calcul présentant des cas particuliers lors de leur exécution, tels que le lancement d'exceptions.

20. Procédé de sécurisation d'instructions d'un objet portable électronique selon l'une quelconque des revendications 8 ou 18, caractérisé en ce que la fonction booléenne Alert s'évalue en VRAI pour au moins un des types suivants d'instructions:

- les instructions de test et/ou
- les instructions émettant de l'information vers l'extérieur par un moyen de communication et/ou
- les instructions modifiant le contenu de la mémoire non-volatile et/ou

- les instructions de calcul présentant des cas particuliers lors de leur exécution, tels que le lancement d'exceptions.

21. Procédé de sécurisation d'instructions d'un objet portable électronique selon l'une quelconque des revendications 8 ou 18, caractérisé en ce que la fonction booléenne Alert s'évalue en VRAI pour au moins un des types suivants d'instructions, si au moins une des données d'entrée est secrète, et FAUX si toutes les données testées sont publiques:
    - les instructions de test et/ou
    - les instructions émettant de l'information vers l'extérieur par un moyen de communication et/ou
    - les instructions modifiant le contenu de la mémoire non-volatile et/ou
    - les instructions de calcul présentant des cas particuliers lors de leur exécution, tels que le lancement d'exceptions.

22. Procédé de sécurisation d'instructions d'un objet portable électronique selon l'une quelconque des revendications 7 à 8 ou 17 à 18, caractérisé en ce que l'ensemble des niveaux de sécurité Φ utilisé lors de l'exécution d'un programme P est indiqué par

la valeur d'une fonction φ, telle que, pour toute donnée u utilisée par le programme, $\varphi(u)=0$ désigne le fait que u est publique et $\varphi(u)=1$ désigne le fait que u est privée, et telle que, pour toute donnée v résultant de l'exécution d'une instruction du programme P, $\varphi(v)=1$ si au moins une des données d'entrée de l'instruction est privée, et sinon $\varphi(v)=0$.

23. Procédé de sécurisation d'instructions d'un objet portable électronique selon la revendication 22, caractérisé en ce que les valeurs de la fonction φ sont calculées au moyen d'une mise en œuvre matérielle d'une fonction « OU logique » réalisée sur les valeurs de la fonction φ pour les données d'entrée des instructions.

24. Procédé de sécurisation d'instructions d'un objet portable électronique selon l'une quelconque des revendications 1 à 23, caractérisé en ce que les fonctions de hachage $HASH_1$, $HASH_2$ et $HASH_3$ sont identiques.

25. Objet électronique caractérisé en ce qu'il met en œuvre l'une quelconque des revendications 1 à 24.

INS_i	effet sur i	effet sur RAM	effet sur ST	effet sur s
inc	$i \leftarrow (i+1)$	aucun	$ST[s] \leftarrow (ST[s]+1)$	aucun
dec	$i \leftarrow (i+1)$	aucun	$ST[s] \leftarrow (ST[s]-1)$	aucun
pop	$i \leftarrow (i+1)$	aucun	$ST[s] \leftarrow$ undef	$s \leftarrow (s-1)$
push0	$i \leftarrow (i+1)$	aucun	$ST[s+1] \leftarrow 0$	$s \leftarrow (s+1)$
load x	$i \leftarrow (i+1)$	aucun	$ST[s+1] \leftarrow RAM[x]$	$s \leftarrow (s+1)$
load IO	$i \leftarrow (i+1)$	aucun	$ST[s+1] \leftarrow IO$	$s \leftarrow (s+1)$
load RNG	$i \leftarrow (i+1)$	aucun	$ST[s+1] \leftarrow RNG$	$s \leftarrow (s+1)$
store x	$i \leftarrow (i+1)$	$RAM[x] \leftarrow ST[s]$	$ST[s] \leftarrow$ undef	$s \leftarrow (s-1)$
store IO	$i \leftarrow (i+1)$	$IO \leftarrow ST[s]$	$ST[s] \leftarrow$ undef	$s \leftarrow (s-1)$
if L	if $ST[s]=0$ then $i \leftarrow (i+1)$ if $ST[s] \neq 0$ then $i \leftarrow L$	aucun	$ST[s] \leftarrow$ undef	$s \leftarrow (s-1)$
goto L	$i \leftarrow L$	aucun	aucun	aucun
xor	$i \leftarrow (i+1)$	aucun	$ST[s-1] \leftarrow ST[s-1] \oplus ST[s]$ $ST[s] \leftarrow$ undef	$s \leftarrow (s-1)$
mul	$i \leftarrow (i+1)$	aucun	$\alpha \stackrel{def}{=} ST[s-1] \times ST[s]$ $ST[s-1] \leftarrow \alpha \bmod 256$ $ST[s] \leftarrow \alpha$ div 256	aucun
div	If $ST[s]=0$ then i ← AdExcDiv if $ST[s] \neq 0$ then $i \leftarrow (i+1)$	aucun	Si (ST[s-1] ≠ 0) faire $ST[s] \leftarrow ST[s] / ST[s-1]$	aucun
		effet sur NVM		
getstatic x	$i \leftarrow (i+1)$	aucun	$ST[s+1] \leftarrow NVM[x]$	$s \leftarrow (s+1)$
Putstatic x	$i \leftarrow (i+1)$	$NVM[x] \leftarrow ST[s]$	$ST[s] \leftarrow$ undef	$s \leftarrow (s-1)$

FIGURE 1

0.	Le XμP initialise $i \leftarrow 1$
1.	Le XμP demande au XT l'ectoinstruction numéro i
2.	Le XT envoi au INSi au XμP
3.	Le XμP exécute INSi
4.	Aller à l'étape 1.

FIGURE 2

INS_i	**Alert = VRAI si**
If L	$\varphi(ST[s])=1$
div	$\varphi(ST[s])=1$
store IO	$\varphi(ST[s])=1$
putstatic x	$\varphi(NVM[x])=1$

FIGURE 3

INS_i	**Alert = VRAI si**
If L	$\varphi(ST[s])=1$
Div	$\varphi(ST[s])=1$
store IO	$\varphi(ST[s])=1$
putstatic x	Toujours

FIGURE 4

INS_i	Effet sur Φ
inc	Aucun
dec	Aucun
pop	$\varphi(ST[s]) \leftarrow 0$
push0	$\varphi(ST[s+1]) \leftarrow 0$
load *x*	$\varphi(ST[s+1]) \leftarrow \varphi(RAM[x])$
load RNG	$\varphi(ST[s+1]) \leftarrow 1$
store *x*	$\varphi(RAM[x]) \leftarrow \varphi(ST[s])$ $\varphi(ST[s]) \leftarrow 0$
load IO	$\varphi(ST[s+1]) \leftarrow 0$
store IO	$\varphi(ST[s]) \leftarrow 0$
if *L*	$\varphi(ST[s]) \leftarrow 0$
goto *L*	Aucun
xor	$\varphi(ST[s-1]) \leftarrow \varphi(ST[s-1]) V \varphi(ST[s])$ $\varphi(ST[s]) \leftarrow 0$
mul	$\varphi(ST[s]), \varphi(ST[s-1]) \leftarrow \varphi(ST[s-1]) V \varphi(ST[s])$
div	$\varphi(ST[s]), \varphi(ST[s-1]) \leftarrow \varphi(ST[s-1]) V \varphi(ST[s])$
getstatic *x*	$\varphi(ST[s+1]) \leftarrow \varphi(NVM[x])$
putstatic *x*	$\varphi(NVM[x]) \leftarrow \varphi(ST[x])$ $\varphi(ST[s]) \leftarrow 0$

FIGURE 5

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP2005/050828

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L9/32 G07F7/10 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L G07F G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, PAJ, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 816 970 A (SUN MICROSYSTEMS, INC) 7 January 1998 (1998-01-07) column 2, line 12 - line 55 column 4, line 30 - column 6, line 36	1-25
A	EP 1 369 764 A (MICROSOFT CORPORATION) 10 December 2003 (2003-12-10) abstract column 2, line 55 - column 4, line 25 column 8, line 53 - column 11, line 15	1-25
A	SE 517 116 C2 (TELEFONAKTIEBOLAGET L M ERICSSON) 16 April 2002 (2002-04-16) abstract	1-25
P,A	-& US 2004/103316 A1 (GEHRMANN CHRISTIAN) 27 May 2004 (2004-05-27) figure 2 paragraphs '0029! - '0050!	1-25

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 June 2005	23/06/2005

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bec, T

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP2005/050828

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	TUAL J-P: "MASSC: A GENERIC ARCHITECTURE FOR MULTIAPPLICATION SMART CARDS" IEEE MICRO, IEEE INC. NEW YORK, US, vol. 19, no. 5, September 1999 (1999-09), pages 52-61, XP000862509 ISSN: 0272-1732 pages 53,55-59	1-25

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP2005/050828

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0816970 A	07-01-1998	US 6138236 A	24-10-2000
		EP 0816970 A2	07-01-1998
EP 1369764 A	10-12-2003	US 2003229777 A1	11-12-2003
		AU 2003204376 A1	08-01-2004
		CN 1469238 A	21-01-2004
		EP 1369764 A2	10-12-2003
		JP 2004013905 A	15-01-2004
SE 517116 C2	16-04-2002	AU 7117701 A	25-02-2002
		CN 1446418 A	01-10-2003
		JP 2004507156 T	04-03-2004
		SE 0002962 A	12-02-2002
		WO 0215466 A1	21-02-2002
		US 2004103316 A1	27-05-2004
US 2004103316 A1	27-05-2004	SE 517116 C2	16-04-2002
		AU 7117701 A	25-02-2002
		CN 1446418 A	01-10-2003
		JP 2004507156 T	04-03-2004
		SE 0002962 A	12-02-2002
		WO 0215466 A1	21-02-2002

INTERNATIONALER RECHERCHENBERICHT

Demande Internationale No
PCT/EP2005/050828

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
CIB 7 H04L9/32 G07F7/10 G06F1/00

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
CIB 7 H04L G07F G06F

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si réalisable, termes de recherche utilisés)
EPO-Internal, PAJ, WPI Data

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie *	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	EP 0 816 970 A (SUN MICROSYSTEMS, INC) 7 janvier 1998 (1998-01-07) colonne 2, ligne 12 - ligne 55 colonne 4, ligne 30 - colonne 6, ligne 36	1-25
A	EP 1 369 764 A (MICROSOFT CORPORATION) 10 décembre 2003 (2003-12-10) abrégé colonne 2, ligne 55 - colonne 4, ligne 25 colonne 8, ligne 53 - colonne 11, ligne 15	1-25
A	SE 517 116 C2 (TELEFONAKTIEBOLAGET L M ERICSSON) 16 avril 2002 (2002-04-16) abrégé	1-25
P,A	-& US 2004/103316 A1 (GEHRMANN CHRISTIAN) 27 mai 2004 (2004-05-27) figure 2 alinéas '0029! - '0050!	1-25

-/--

[X] Voir la suite du cadre C pour la fin de la liste des documents

[X] Les documents de familles de brevets sont indiqués en annexe

* Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

Date à laquelle la recherche internationale a été effectivement achevée	Date d'expédition du présent rapport de recherche internationale
16 juin 2005	23/06/2005

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bec, T

Formulaire PCT/ISA/210 (deuxième feuille) (janvier 2004)

INTERNATIONALER RECHERCHENBERICHT

Demande Internationale No
PCT/EP2005/050828

C.(suite) DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie *	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	TUAL J-P: "MASSC: A GENERIC ARCHITECTURE FOR MULTIAPPLICATION SMART CARDS" IEEE MICRO, IEEE INC. NEW YORK, US, vol. 19, no. 5, septembre 1999 (1999-09), pages 52-61, XP000862509 ISSN: 0272-1732 pages 53,55-59	1-25

INTERNATIONALER RECHERCHENBERICHT

Demande Internationale No
PCT/EP2005/050828

Document brevet cité au rapport de recherche		Date de publication	Membre(s) de la famille de brevet(s)		Date de publication
EP 0816970	A	07-01-1998	US	6138236 A	24-10-2000
			EP	0816970 A2	07-01-1998
EP 1369764	A	10-12-2003	US	2003229777 A1	11-12-2003
			AU	2003204376 A1	08-01-2004
			CN	1469238 A	21-01-2004
			EP	1369764 A2	10-12-2003
			JP	2004013905 A	15-01-2004
SE 517116	C2	16-04-2002	AU	7117701 A	25-02-2002
			CN	1446418 A	01-10-2003
			JP	2004507156 T	04-03-2004
			SE	0002962 A	12-02-2002
			WO	0215466 A1	21-02-2002
			US	2004103316 A1	27-05-2004
US 2004103316	A1	27-05-2004	SE	517116 C2	16-04-2002
			AU	7117701 A	25-02-2002
			CN	1446418 A	01-10-2003
			JP	2004507156 T	04-03-2004
			SE	0002962 A	12-02-2002
			WO	0215466 A1	21-02-2002